11-2009

The Discrete Fourier Transform, Part 4: Spectral Leakage

Douglas A. Lyon
Fairfield University, dlyon@fairfield.edu

Published Citation

Douglas Lyon, "The Discrete Fourier Transform, Part 4: Spectral Leakage", Journal of Object Technology, Volume 8, no. 7 (November 2009), pp. 23-34

JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©JOT, 2009

Vol. 8, No. 7, November-December 2009

The Discrete Fourier Transform, Part 4: Spectral Leakage

By Douglas Lyon

Abstract

This paper is part 4 in a series of papers about the Discrete Fourier Transform (DFT) and the Inverse Discrete Fourier Transform (IDFT). The focus of this paper is on spectral leakage. Spectral leakage applies to all forms of DFT, including the FFT (Fast Fourier Transform) and the IFFT (Inverse Fast Fourier Transform). We demonstrate an approach to mitigating spectral leakage based on windowing. Windowing temporally isolates the Short-Time Fourier Transform (STFT) in order to amplitude modulate the input signal. This requires that we know the extent, of the event in the input signal and that we have enough samples to yield a sufficient spectral resolution for our application.

This report is a part of project Fenestratus, from the skunk-works of DocJava, Inc. Fenestratus comes from the Latin and means "to furnish with windows".

1 INTRODUCTION TO FREQUENCY QUANTIZATION AND LEAKAGE

The DFT is a data processing tool whose input is a digital signal and whose output is its digital spectral analysis. The discrete nature of the DFT yields intrinsic quantization errors and sampling artifacts. This papers addresses the intrinsic errors introduced by the DFT and discusses windowing as a means of error mitigation.

Suppose that the sampling rate for digitization is selected to be twice the highest harmonic in the input signal and is denoted by:

$$f_s = \text{The sampling rate} \qquad (1).$$

The basic idea of (1) is that the DFT spectral analysis will be constrained to range between zero and one-half of the sampling rate. Thus, if the sampling rate is 8,000 Hz, the DFT output will measure frequencies from 0 Hz to 4,000 Hz. Thus, the bandwidth of the DFT is proportional to sample rate.

The sampling period, Δt, is computed from the sampling rate using

$$\Delta t = 1 / f_s \qquad (2).$$

The kth array element in the frequency domain is referred to as a frequency bucket or *bin*. Thus we can relate the frequency of bucket k to the frequency of the input signal using:

$$f_k = \frac{k}{N\Delta t}$$

where

$k =$ bucket number (3).

$N =$ number of samples

$f_k =$ frequency of bucket k

Equation (3) can also be used to obtain the *frequency quantum.* The frequency quantum is the change in frequency that is represented by two sequential bucket numbers. For example, suppose that samples, $N = 2048$, $k = 1$ and the sample rate is 8000 Hz. Then frequency quantum is 8000/2048 = 3.9 Hz. For $k = 0$, we have a D.C. component that has zero Hz. Thus, the step between bucket 0 and bucket 1 represents a quantization in frequency.

The central point of the spectrum is given by $N/2 = 2048/2 = 1024$. By (3) we compute the highest frequency that may be represented by 2048 samples (with 8khz sampling rate) as (8000/2048) *1024 = 4000 Hz. Also, we may solve for any sample using:

$$k = Nf_k / f_s \quad (4).$$

Equation (4) allows us to make two basic assertions about trade-offs in the DFT:

1. Frequency resolution is inversely proportional to sample rate, for a given number of samples.
2. For a given sample rate, frequency resolution is proportional to the number of samples.

The Heisenberg uncertainty principle states that certain pairs of physical properties cannot be known to arbitrary precision. In the case of quantum physics, the properties are position and momentum. In the case of the DFT, the properties are frequency and time. Thus, we need more samples (that is more time) to get better frequency precision. However, this causes our ability to localize the event (precision in time) to suffer. The converse is also true. As we try to isolate the event in time, the number of samples has declined and so has the precision in frequency. In fact, from (3) we can see that the frequency precision is proportional to the number of samples. Further, we can compute the frequency for any bucket.

For example, for a 400 Hz waveform we expect the maximum amplitude to occur at $k = Nf_k / f_s = 2048 * 400 / 8000 = 102.4$. Each bucket corresponds to a frequency, not a range of frequencies. And a bucket number is always an integer. How can the DFT represent energy at k = 102.4? The answer is that the energy *leaks* between buckets k=102, and 103 and thus the name *spectral leakage*.

The problem with spectral leakage is that it is different for different frequencies. For example, at 390.625 Hz, $k = 2048*390.625/8000 = 100$, *exactly*. Thus, there is no spectral leakage for some frequencies, where there may be a great deal of spectral leakage for others. This is the frequency-domain rationale for spectral leakage.

The time-domain rationale for spectral leakage is that, for a waveform that is not periodic in time, the temporal effect of the sample window becomes visible in the Fourier transform [Walker]. Further, if an input signal is finite and the window

surrounding the input signal is rectangular then the Fourier transform of the rectangular window is given by the *sinc function:* $\operatorname{sinc} x = \sin(x)/x$.

2 WINDOWING

"Windowing" amplitude modulates the input signal so that the spectral leakage is evened out (spreading on-bucket signals more and off-bucket signals less). Thus, windowing reduces the amplitude of the samples at the beginning and end of the window, altering leakage. Windowing is implemented by multiplying the input signal with a *windowing function*. An input signal can be of any number of dimensions and can be complex. For complex-values a complex multiplication is required. For the purpose of simplicity, we assume that the input signal is 1D and real-valued.

Windowing functions are also called *tapering* functions or *apodization* functions. Apodization is from the Greek and literally means, „removing the feet". It is the technical term for changing the shape of a signal by the use of a function that has a zero-value outside of a selected interval. It involves using a tapering function to smooth out the transitions.

There are many common windowing functions. For example, the Hann filter (also called the hanning filter) is named after the Vienese metrologist, Julius Ferdinand von Hann (1839-1921). The Hann window is given by:

$$w_j = \frac{1}{2}\left[1 - \cos\left(\frac{2\pi j}{N-1}\right)\right] \quad j \in [0...N-1] \tag{5}$$

The window falls off at -18 dB per octave. Figure 1 shows the Hann window.

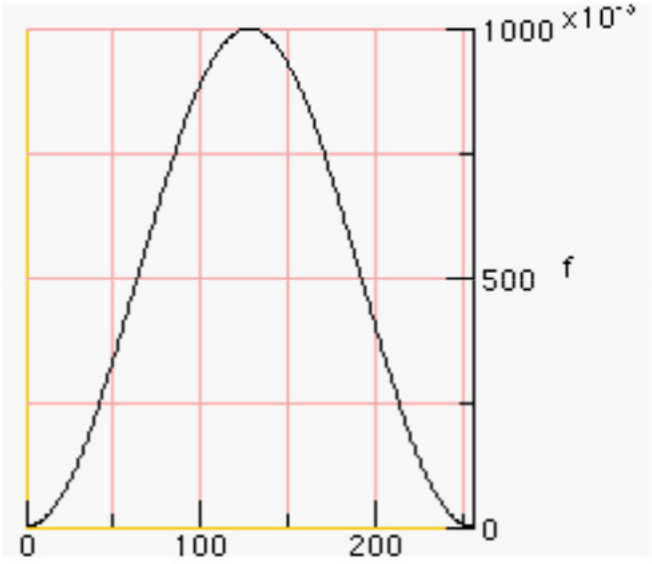

Figure 1. The Hann window

The following code graphs the Hann window:

```
public void makeHann() {
 double window[];
      window = makeHann(256);
      Graph.graph(window,
                                "The Hanning window","f");
 }
```

The method returns an array of double.

```
 public double[] makeHann(int n) {
      double window[] = new double[n];
      double arg = 2.0 * Math.PI/ (n - 1);
      for (int i = 0; i < n; i++)
            window[i] = 0.5 - 0.5 * Math.cos(arg*i);
      return window;
}
```

To apply the window to the samples, we have devised a *windowArray* method:

```
public void multHann() {
      double[] r_d = getTruncatedDoubleData();
      double[] window = makeHanning(r_d.length);

      windowAudio(r_d, window, "Hann");
 }

 public void windowArray(double window[],double r_d[]) {
      for (int i = 0; i < window.length; i++) {
            r_d[i] *= window[i];
      }
 }
```

The primary difference between one window and the next is the way it tapers off at the ends of the samples. One window, called the *Bartlett* window, is a linear window that is described by:

$$w_j = \begin{cases} \dfrac{2j}{N-1} & j \in \left[0 \ldots \dfrac{N-1}{2}\right] \\ 2 - \dfrac{2j}{N-1} & j \in \left[\dfrac{N-1}{2} \ldots N-1\right] \end{cases} \tag{6}$$

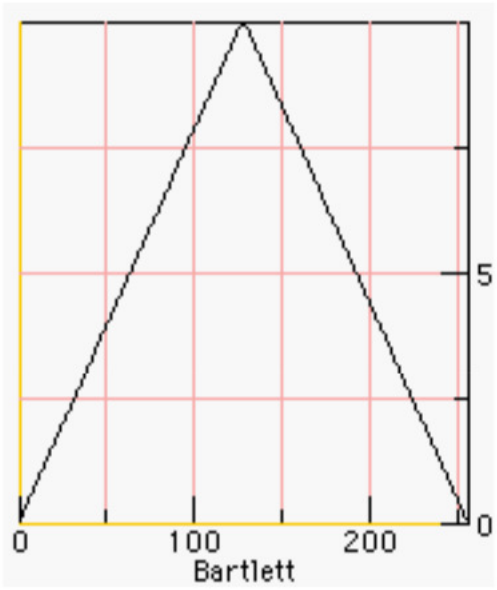

Figure 2. The Bartlett Window

Figure 2 shows a graph of a Bartlett window [Bartlett]. The Bartlett window is computed using:

```
public double[] makeBartlett(int n) {
      double window[] = new double[n];
      double a = 2.0 /  (n - 1);
      for (int i = 0; i < n/2; i++)
            window[i] = i * a;
      for (int i = n/2; i < n; i++)
            window[i] = 2.0 - i * a;
      return window;
 }
```

3 A VANITY WINDOW

In this section I create a window, just for fun. Lets call it the *Lyon window*, first described in [Lyon 1997]. Suppose we start with constraints on a window, then use these constraints to create a polynomial. We propose a window that is zero at the sample end-points and also has zero first and second derivatives at the end-points and the center. Such a window may be formulated with two fifth-order polynomials (two quintics). We present some Maple code for deriving a quintic-based window.

Maple is a symbolic manipulator that can help with some math problems. In Maple, we let dy and ddy be the first and second derivatives of the polynomial with respect to the dilation parameter u. The dilation parameter will be varied between zero and one, inclusive. The amplitude of the qunitic will vary from y0 to y1. Maple's optimize routine can output a procedure for computing quintics in minimum CPU time. What follows is a Maple procedure for generating the Lyon window. See [Lyon 91] for an application of the quintic to maneuvering.

```
restart;y:=a5*u^5+a4*u^4+a3*u^3+a2*u^2+a1*u+a0:
> dy:=diff(y,u):
> ddy:=diff(dy,u):
> a0:=y0:
> a1:=solve(y1=subs(u=1,y),a1):
> a2:=solve(0=subs(u=0,dy),a2):
> a3:=solve(0=subs(u=1,dy),a3):
> a4:=solve(0=subs(u=0,ddy),a4):
> a5:=solve(0=subs(u=1,ddy),a5):
> readlib(C):
> C(y,optimized);
      t2 = u*u;
      t3 = t2*t2;
      t11 = (6.0*y1-6.0*y0)*t3*u+(-15.0*y1+15.0*y0)*t3+(10.0*y1-
10.0*y0)*t2*u+y0;
```

We compute the quintic using:

```
public double y5(double y0, double y1, double u) {
     double t2 = u*u;
     double t3 = t2*t2;
     return
            (6 * y1 - 6 * y0) * t3 * u +
            (-15 * y1 + 15 * y0) * t3 +
            (10 * y1 - 10 * y0) * t2 * u + y0;
 }
```

The equations for the Lyon window is given by:

$$w_j = \begin{cases} 6u^5 - 15u^4 + 10u^3 & u = \dfrac{2j}{N-1}, j \in \left[0 \ldots \dfrac{N-1}{2}\right] \\ -6u^5 + 15u^4 - 10u^3 + 1 & u = \dfrac{2j+1-N}{N-1}, \; j \in \left[\dfrac{N-1}{2} \ldots N-1\right] \end{cases} \tag{7}$$

The easier way to compute the dilation parameter, u, is by incrementing it by 2/N as in:

```
public double[] makeLyon(int n) {
     double window[] = new double[n];

     double u = 0.0;
     double du = 2.0/n;
     for (int i = 0; i < n/2; i++) {
            window[i] = y5(0,1.0,u);
            u += du;
     }
     u=0;

     for (int i = n/2; i < n; i++) {
            window[i] = y5(1.0,0.0,u);
            u += du;
     }
     return window;

}
```

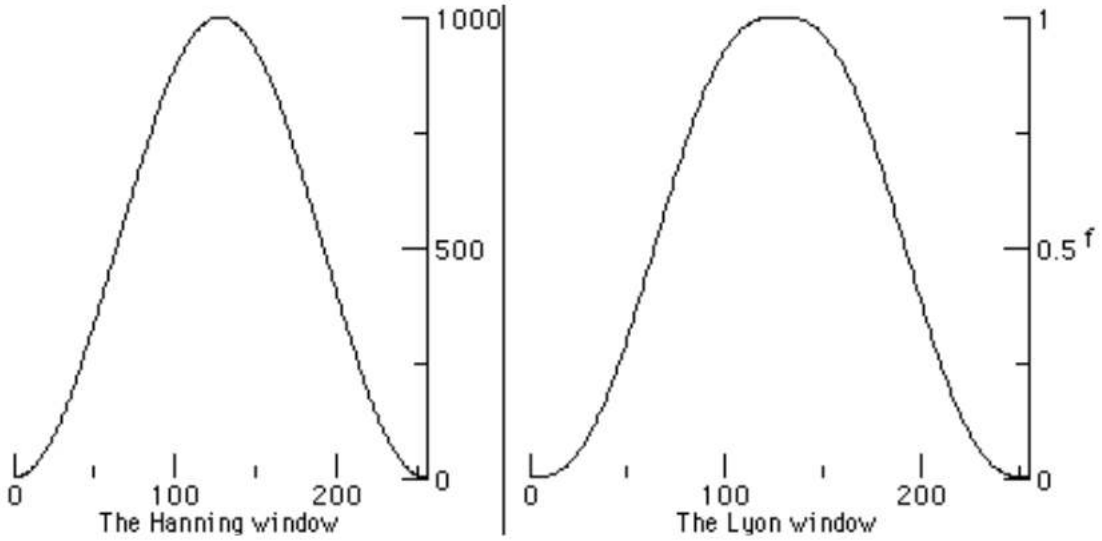

Figure 3. The Lyon and Hann windows compared

The Lyon and Hann windows are shown in Figure 3. The Lyon window has a flat head (in German, they say "flachem Kopf"), increasing the number of unattentuated samples at the centroid of the event. The flachem kopf windows have zero first and second derivatives at their extrema, thus reducing distortion and "clicks".

4 THE HI-PASS FILTER

One very simple way to design a hi-pass filter is to take an FFT on the input samples, then multiply the spectrum by the filter envelope. For example, to make a hi-pass filter, we may zero out the low frequencies using a rectangular pass-band function, like the one shown in Figure 4.

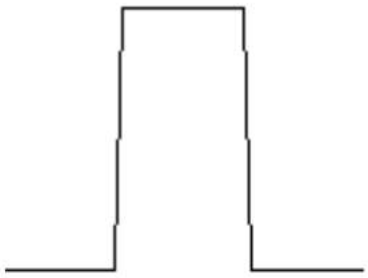

Figure 4. A passband shown spectral harmonics to admit the PSD

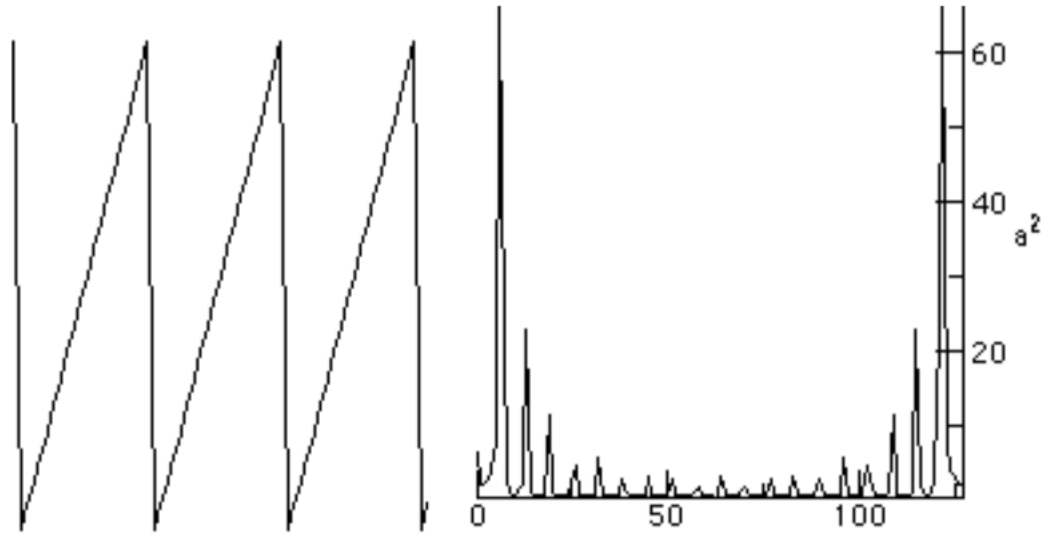

Figure 5. The Saw wave and its PSD

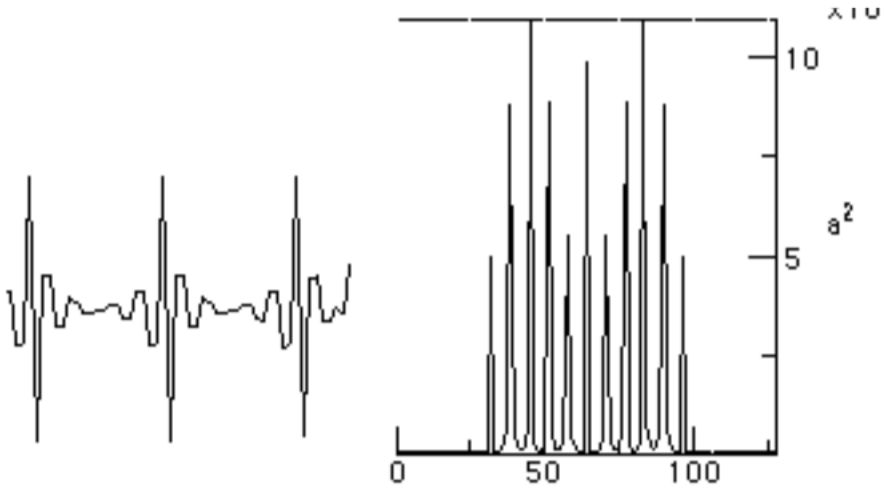

Figure 6. The Hi-pass filtered Saw wave and its PSD

For the PSD depicted in this chapter, the higher frequencies are toward the center of the graph. For the determination of the performance of the various windows, we follow [Embree] and plot the windowed waveform along side the log (in dB) of the PSD. We use the dB log and zoom into the first 200 samples of the PSD to make smaller details clear. We compute the graph using:

```
public void graphPSD() {
     double psd[] = fftInstance.computePSD();
     double shortPsd[] = new double[200];
     for (int i =0 ; i < shortPsd.length; i++) {
      shortPsd[i] = 10*Math.log(Math.sqrt(psd[i]));
     }
     Graph.graph(shortPsd, "spectral mag","a dB");
 }
```

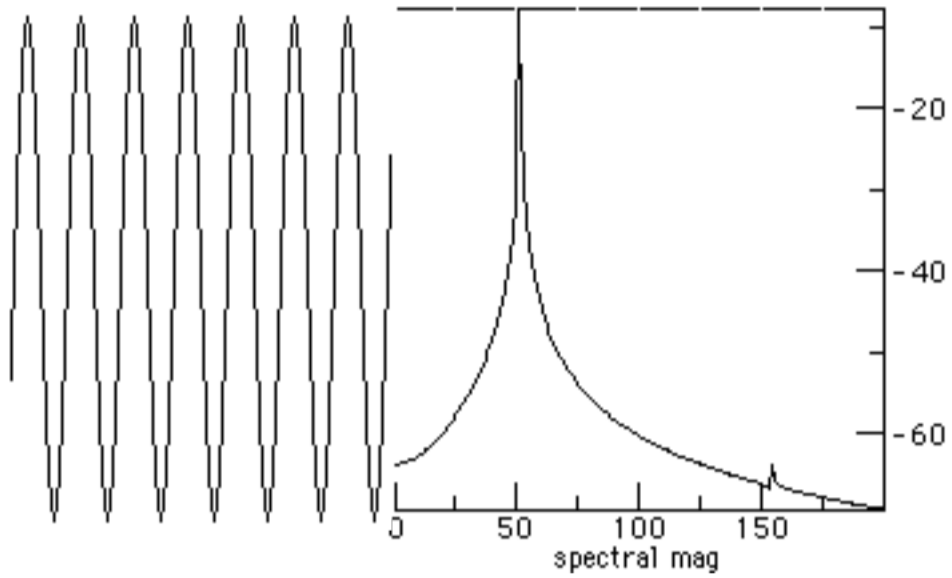

Figure 7. A Rectangular window and spectral dB log.

Figure 7 shows a sine wave with a rectangular window. The sine wave is 400 Hz with 8000 Hz sampling. This is often expressed as a relative frequency of 400/8000 = 0.05.

Figure 8 shown the Hann window with the spectral log magnitude.

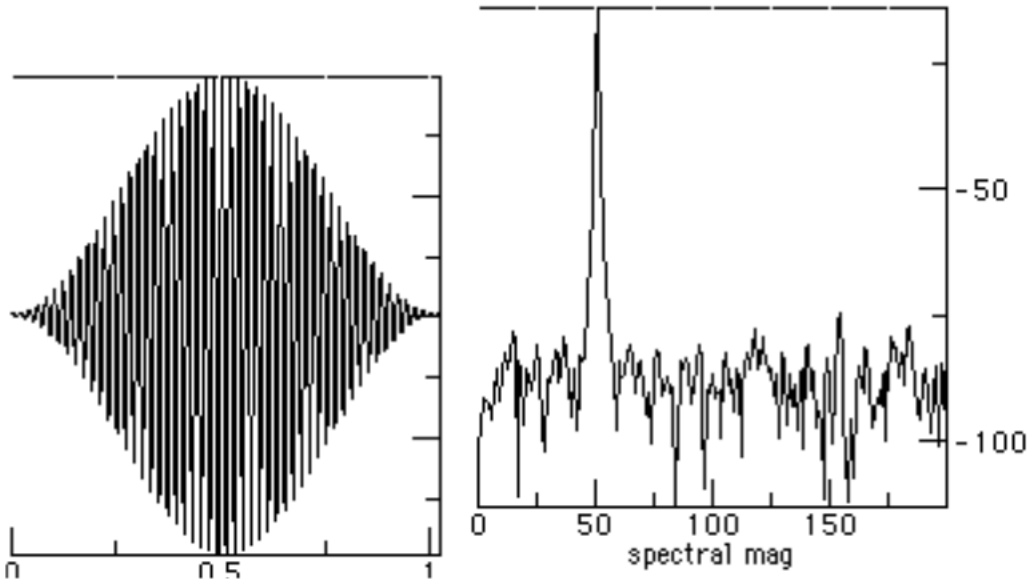

Figure 8. The Hann windowed data with the FFT result.

The triangular windowed input of the Bartlett window is shown in Figure 9, along with the PSD.

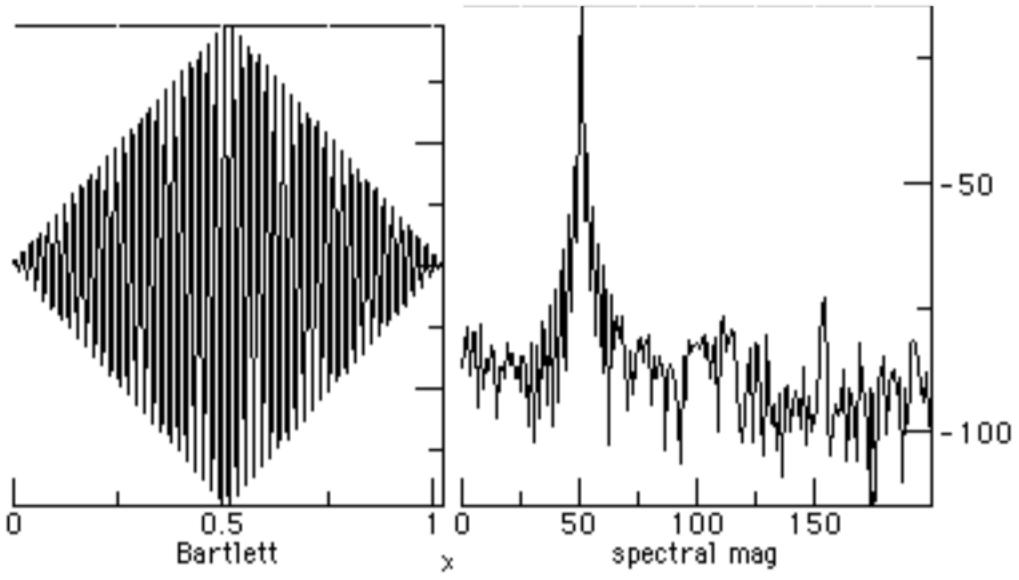

Figure 9. The Bartlett windowed data and the PSD

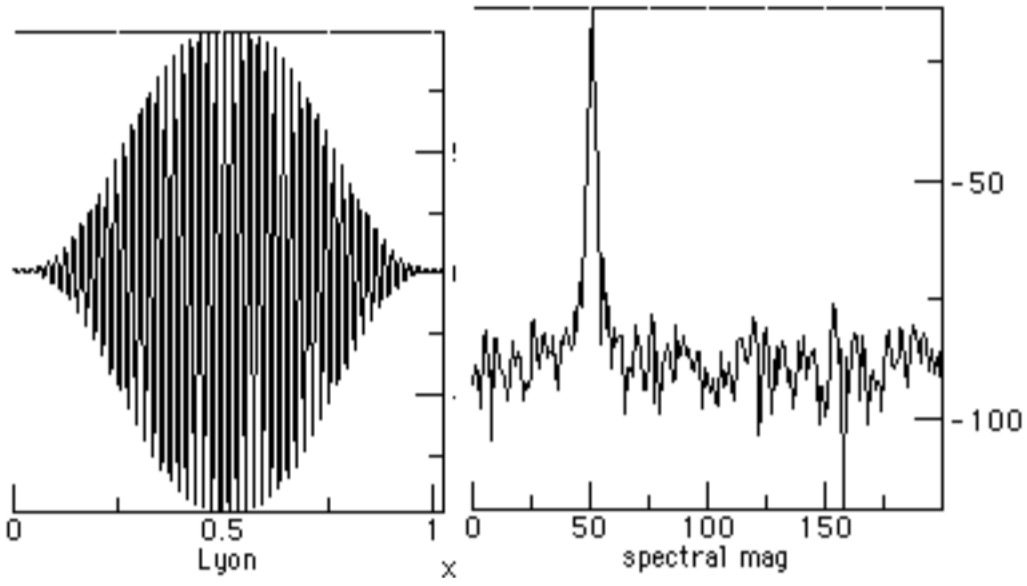

Figure 10. The Lyon window and PSD

Figures 9 and 10 show that the Lyon window has somewhat better side lobe performance than the Bartlett window (lower spectral leakage). The main lobe in the Lyon window is narrower than the Bartlett window (lower noise bandwidth).

Figure 11 shows the spectra for the Lyon and Hann windows on the left and right, for comparison.

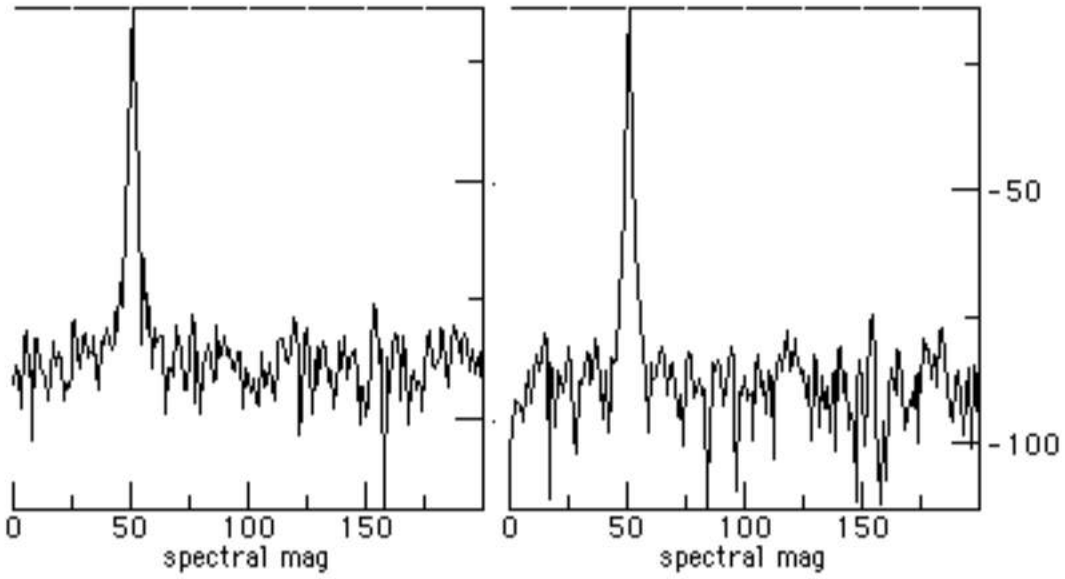

Figure 11. Spectra of Lyon vs. Hann windows

The Lyon and Hann windows appear to be spectrally close. The Lyon window appears to have a few dB better side lobe performance over the Hann window. The main lobe is slightly wider, however, and so has a slightly larger equivalent noise bandwidth [Mitra].

5 SUMMARY

We have seen windowing implemented as the amplitude modulation of a signal by using a finite duration function. The primary goal of windowing is to force the resulting signal to be zero outside of the window range, controlling the transition so that it is not too abrupt.

The derivation of the Lyon window, using Maple, was shown in [Lyon 1997]. In the past the quintic was viewed as a curvature controlled trajectory for maneuvering a car [Lyon 90]. We make no claim as to the suitability of the Lyon window for signal processing since the window has not been thoroughly investigated and this poses a topic of future research.

There are several figures of merit for the various windows. These are explored in detail by [Harris]. While there is a compelling argument for making using of a figure of merit to select a window, there is an equally compelling argument that the figure of merit itself may be arbitrary. The question of which figure of merit to use is left open.

REFERENCES

[Arons] "SpeechSkimmer: A System for Interactively Skimming Recorded Speech", by Barry Arons. *ACM Transactions on Computer-Human Interaction* 4:1, pps. 3-38, 1997.

[Bartlett] Bartlett, M. S. "Periodogram Analysis and Continuous Spectra", *Biometrika 37*, 1-16, 1950.

[Embree] *C Language Algorithms for Digitial Signal Processing*,by Paul M. Embree and Bruce Kimble, Prentice Hall, Englewood Cliffs, NJ, 1991.

[Harris] Fredric J. Harris, "On the use of Windows for Harmonic Analysis with the Discrete Fourier Transform", Proceedings of the IEEE, V66N1, Jan. 1978, pp. 51-83.

[Lyon 90] "Ad-Hoc and Derived Parking Curves", by Douglas Lyon, SPIE - International Society for Optical Engineering, Boston MA, November 8, 1990.

[Lyon 91] *Parallel Parking with Nonholonomic Constraints.* PhD thesis, by Douglas Lyon, Computer and Systems Engineering Department, RPI, Troy, NY, 12181. Available at http://www.docjava.com, 1991

[Lyon 97] *Java Digital Signal Processing*, Douglas A. Lyon and H. Rao, M&T Press (an imprint of Henry Holt). November 1997.

[Mitra] *Handbook for Digital Signal Processing*, Sanjit K. Mitra and James F. Kaiser, John Wiley & Sons, NY, NY, 1993.

[Walker] *Fast Fourier Transforms*, James S. Walker. CRC Press, NY, NY, 1996.

About the author

Douglas A. Lyon (M'89-SM'00) received the Ph.D., M.S. and B.S. degrees in computer and systems engineering from Rensselaer Polytechnic Institute (1991, 1985 and 1983). Dr. Lyon has worked at AT&T Bell Laboratories at Murray Hill, NJ and the Jet Propulsion Laboratory at the California Institute of Technology, Pasadena, CA. He is currently the co-director of the Electrical and Computer Engineering program at Fairfield University, in Fairfield CT, a senior member of the IEEE and President of DocJava, Inc., a consulting firm in Connecticut. Dr. Lyon has authored or co-authored three books (Java, Digital Signal Processing, Image Processing in Java and Java for Programmers). He has authored over 40 journal publications. Email: lyon@docjava.com. Web: http://www.DocJava.com.